Journal of King Saud University – Computer and Information Sciences (2012) **24**, 137–144

King Saud University

**Journal of King Saud University – Computer and Information Sciences**

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

# Management of interactive business processes in decentralized service infrastructures through event processing

**Thomas Schlegel [a,*], Krešimir Vidačković [b], Sebastian Dusch [c], Ronny Seiger [a]**

[a] *SEUS Group, Technische Universitaet Dresden, Noethnitzer Str. 46, 01062 Dresden, Germany*
[b] *Fraunhofer IAO, Competence Center Electronic Business, Nobelstr. 12, 70569 Stuttgart, Germany*
[c] *Institute for Visualization and Interactive Systems (VIS), Universitaet Stuttgart, Universitaetsstr. 38, 70569 Stuttgart, Germany*

Received 15 June 2011; revised 1 March 2012; accepted 14 March 2012
Available online 5 April 2012

**KEYWORDS**

Service platform management;
Complex event processing;
Decentralized processes;
Web-service orchestration;
Business process monitoring;
Multi-engine environments

**Abstract** Several independent service providers often form decentralized service infrastructures. However, efficient management and collaboration is impossible, if the execution engines are not properly connected. The decentralized approach requires an infrastructure that connects the engines and additionally provides management access to the infrastructure and processes executed. When business processes are executed using multiple process execution engines, monitoring and management of these processes become impossible using standard tools. Therefore, management by an organization providing a common platform integrating the different service providers is required. In this paper, we present an approach and an implementation of such a service platform, using a complex event processing (CEP) engine to integrate different process execution engines and other applications. In such a setting, it becomes even irrelevant if process execution is based on the Web Service Business Process Execution Language (WS-BPEL) or the executable Business Process Model and Notation (BPMN). As being able to interact with such processes and running services is crucial in such an infrastructure, we provide a concept for creating ad-hoc user interactions on a monitoring dashboard, which allows platform managers as well as stakeholders in the processes to interact with the platform and the processes executed – independent of their execution context.


## 1. Introduction

### *1.1. A new complexity in service infrastructures*

Service infrastructures become more and more complex, as different services emerge each day, which are again incorporated in processes that use them in order to achieve their goals building on the services' functionalities. For example, insurance companies relying on a service-oriented architecture (SOA) will need to monitor their cross-company processes and events that occur in their infrastructure. In virtual enterprises or even fully

* Corresponding author. Tel.: +49 351 463 39177; fax: +49 351 463 38518.
E-mail addresses: thomas.schlegel@tu-dresden.de (T. Schlegel), kresimir.vidackovic@iao.fraunhofer.de (K. Vidačković).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

1319-1578 © 2012 King Saud University. Production and hosting by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jksuci.2012.03.001

decentralized service platforms, monitoring and management of activities, processes and exceptions becomes a key problem.

In such a scenario, a suitable architecture has to support the exchange of information and triggers between different processes as well as a management platform that is capable of monitoring a process and service landscape, also generating interactions with users in different roles (e.g. platform managers and process participants). We present an architectural approach and implementation of such a system, using a complex event processing (CEP) engine to integrate different process execution engines and other applications.

The rest of this work is structured as follows: Section 1.2 introduces a practical use case scenario which will be used throughout the whole paper. Section 2 discusses some basic concepts in the field of process modeling and execution, complex events and interactive control that have been used in our system architecture. Sections 3 and 4 describe our approach and implementation in more detail. Section 5 talks about related work in the field of complex event processing and business process monitoring. Section 6 discusses and evaluates our results and shows starting points for further research.

### *1.2. Use case scenario*

Throughout this paper we will be using a scenario from the insurance domain consisting of simple automated processes to illustrate and explain our concepts and practical implementations. In this context, we assume the existence of a service platform where several providers offer their electronic and humanly executed services for different insurance companies who integrate them into their cross-company business processes. An example is the claim settlement process of property insurances regulating storm damages on roofs of insured houses. The service providers in this case are various roofers, but also providers of electronic services, e.g., a weather information service as well as a fraud detection service using the former to check the notification of a claim. Other electronic services could be a price information service and a checking service which checks the bills of roofers for appropriateness. A simplified claims settlement process contains the following steps:

1. Customer informs insurance company about the storm damage.
2. Insurance company uses fraud detection service to check the notification of claim for fraud.
3. Insurance company searches roofer over the platform.
4. Roofer interacts with the platform by accepting the job.
5. Roofer repairs storm damage and sends bill to insurance company.
6. Insurance company uses checking service to check the bill for appropriateness and pays the bill.

In such cross-company business processes, there are several monitoring tasks to perform in order to make sure that the process is passed through in the desired way. In addition, several explicit and implicit interactions between the process and the participants are required as well. Therefore, such a process is an adequate test scenario for our event-based concept and system implementation.

## 2. Process execution, events and interactive control

### *2.1. Process modeling and execution*

Where automatable IT-based business processes are concerned, commonly the workflows are modeled and executed using the Web Services Business Process Execution Language (WS-BPEL) (Alves et al., 2007) or recently the current version 2.0 of the Business Process Model and Notation (BPMN) (OMG, 2011), which now includes all the necessary means to not only model, but also to execute service-based processes as well (Silver, 2010). WS-BPEL and BPMN 2.0 processes have the advantage of being executable and using web service calls directly within the processes.

Unfortunately, monitoring of processes is not easy, even on only one engine instance running. It currently becomes nearly impossible, if multiple processes are executed on different servers or engines. Therefore, it is crucial to achieve monitoring of decentralized processes in order to manage complex service infrastructures.

### *2.2. Complex event processing*

A suitable way to monitor decentralized processes in complex service infrastructures is the application of an event-driven approach where events from different event sources are analyzed and processed by an event processing engine (Vidačković et al., 2009). In this setting, an event represents any meaningful happening from an internal or external event source (Luckham and Schulte, 2008). These events are routed to the engine either using data streams, e.g. in Stream,[1] or the engine is embedded into an application and receives events directly from an API, e.g. in Esper.[2]

When event aggregations and abstractions are applied on a multitude of events from a so-called event cloud in order to detect hidden and complex relations between them represented by event patterns, the term complex event processing (CEP) is used (Luckham, 2002). Event patterns could contain, for example, temporal, causal or spatial relations between events, conjunctions, disjunctions or negations of events and much more. If such defined event patterns are detected within the incoming event cloud, the CEP engine immediately emits an event as a real-time reaction on this detection (Etzion and Niblett, 2010). Hence, CEP is an appropriate technology for the real-time monitoring of distributed systems or processes with complex relations.

Event-driven architecture (EDA) is an architecture pattern (Chandy and Schulte, 2010) that is often applied in software development, if events are produced and consumed in different parts of a software product. One big advantage of EDA in dynamic and decentralized systems is the fact that producers of events do not need to know the consumers of a produced event and vice versa, so that highly loose coupling is accomplished. The event-driven concept is often implemented using publish-subscribe mechanisms (Mühl et al., 2006). The architecture pattern consists of three parts: an event source produces events, while event channels transmit them to subscribed event sinks. Components in an event-driven architecture may take the role of event sinks and event sources at the same time. This

[1] http://infolab.stanford.edu/stream
[2] http://esper.codehaus.org

is especially the case, when a CEP engine is involved which processes received events from different event sources and reacts on identified event patterns by sending new events to downstream components as a real-time reaction (Bruns and Dunkel, 2010).

Therefore, we are using a CEP engine to enable routing and rule-based interpretation of events between different processes and execution engines. In that way, relevant events from within the processes only need to be sent by the process execution engine to the CEP engine for analysis, while the technology used for process execution itself is not relevant for monitoring purposes. This may hence be any WS-BPEL or BPMN 2.0 engine, if it is only able to send the required types of events to the CEP engine. With regard to service and process infrastructures, complex event processing offers a valuable approach to dispatch information, status changes and actions across multiple processes and platforms (Vidačković et al., 2009).

Fig. 1 shows the graphical illustration of the WS-BPEL process of the fraud detection service described in Section 1.2 as an example. The same process can also be realized in BPMN 2.0 and executed on a BPMN 2.0 engine with the same web service calls and event generations which are transmitted to the CEP engine and analyzed for further processing.

### 2.3. Interaction with service and process infrastructures

Approaches to integrate human activities explicitly in the process have been led to the specification of BPEL4People (Agrawal et al., 2009), which aims at including human actions into a WS-BPEL process. In conjunction with BPEL4People, WS-Human Task (WS-HT) (Agrawal et al., 2007) serves as a basis for including human tasks. WS-HT is based on WSDL, which is used to describe service interfaces, making it possible to also describe human users (roles, timeouts and interactions) as part of the process. In BPMN 2.0, human tasks in business processes are modeled and executed using the user task element.

BPEL4People and WS-HT as well as the BPMN 2.0 user task element serve as a means for modeling and integrating

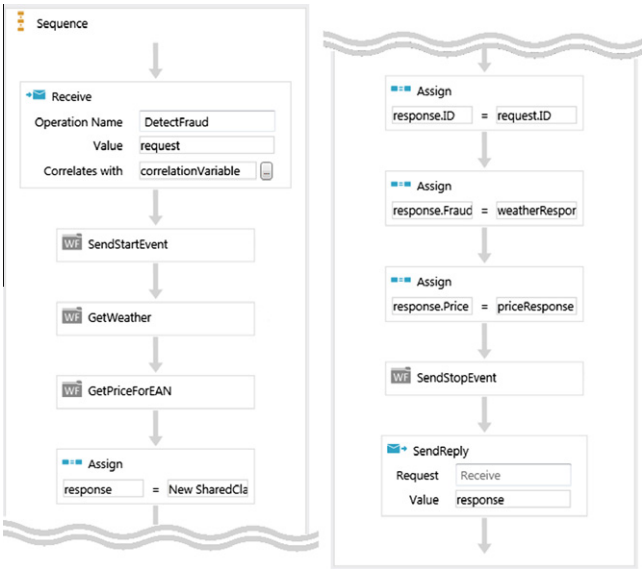


**Figure 1** WS-BPEL process of the fraud detection service as an example.

interactions and generally human tasks into a WS-BPEL process and a BPMN 2.0 process respectively, executed in a process engine. We refer to this kind of interaction as explicit interactions (Schlegel, 2010). Explicit interactions comprise all interactions that are foreseen in the process, i.e. they are expected to occur and have to be accomplished to execute the process. Most approaches for model-based and generative user interfaces use their modeling methods for describing explicit interactions in descriptive models. These models are then transformed to user interfaces, e.g. through multiple model levels like the ones described by the Cameleon reference framework (Calvary et al., 2003). As the semantics can be described on design time of the processes, powerful transformations and mappings can be used to generate user interfaces either through model transformation or model interpretation.

However, not all interactions can be foreseen and modeled before process execution. Therefore, implicit interactions (Schlegel, 2010) form a second category of interactions that is not modeled within the process, but becomes necessary when unforeseen situations, interactions between processes or interactions outside the process (e.g. management of the service or process platform) occur. As no upfront modeling is possible and therefore no model is available, the execution engine has to deal with such interactions on runtime. Implicit interactions often become necessary when data is missing in the process and therefore is requested from the CEP engine or when the CEP engine itself identifies data missing from the events handled. For example, when a shipping process is triggered for routing goods in a company to another place internally, while normally the service only handles shipping to customers, it becomes necessary to specify the internal recipient and adapt the billing information. This can be accomplished through implicit interactions generated by the CEP engine in order to fill the missing "customer" information.

As the technology targets the management and operation of a multi-tenant service platform, interactions are generated for three different groups of stakeholders requiring different handling:

- Customers, who interact with the platform for using services and participating in processes, e.g. filling in data to collaborate with a service or service provider,
- Service providers, who provide and manage their services and processes via the platform and therefore trigger and control the processes through interaction and data delivered by the services, and
- Platform providers, who manage the platform and the processes executed on it and therefore have control over a part of or the complete infrastructure and state of the platform, including triggering and evaluating events and processes.

All the three categories of stakeholders require different access levels, functionalities and interactions, while they all use implicit and explicit interactions for their work. From customers over service providers to platform providers, more and more implicit interactions are used as new tasks and incomplete process settings arise.

### 2.4. Interaction by events

We use events in the CEP engine not only for monitoring purposes, but also for triggering interactions and returning results.

A dashboard is used to show these events and to execute the interaction commands. In this way, the events can be used to trigger implicit interactions as well as explicit interactions. Each process (back-end), and each dashboard (control front-end) can trigger interactions by creating an interaction event (cf. Section 3.1).

Each interaction event carries with it a model-based payload that is able to describe the interaction needed and also the rules and structures of the data created by the interaction. Data consumed and created in an interaction can vary from complex and rule-based entities to simple events without any data payload.

The system conceptually supports many different approaches for generating interactions, ranging from purely client-sided concepts (1) that independently interpret the request to derive a specific interaction in the user interface to a remotely managed interaction, (2) that receive a user interface description which is then only being displayed. For our prototypical implementation we have used a more back-end oriented option, i.e. variant (2), which is based on model fragments in XAML (eXtensible Application Markup Language). Fig. 2 shows a screenshot of the graphical user interface based on XAML. The dialog asking for acceptance of a task is generated from a specific event received from the CEP engine.

## 3. System description

First, we present the basic model used to describe events and transmit them into the system. Then, we describe the system architecture, which allows the usage of multiple WS-BPEL and BPMN 2.0 execution engines that transmit events from within their processes to the complex event processing system. The client application used to monitor, manage and interact with the processes receives events from and also transmits events to the CEP engine.

### 3.1. Event model

The notion of events is the main concept that helps in distributing processes over multiple engines on a platform. It also facilitates integration by enabling decentralized communication, i.e. clients and process engines do not have to know each other, but will, nevertheless, be interconnected by the CEP engine. We use the W3C standard WS-Eventing (Box et al., 2006) to define the contents (body) of a SOAP message that is used for subscribing and unsubscribing to specific events. These events are then also sent to the recipient within the SOAP body. This makes it possible to register applications on events generated by any process execution engine.

Our developed event schema consists of all the events that can be created by the different process execution engines. It describes the properties of each type of event. To allow for an easy use and exchange of the event schema and events, we have used XML schema as definition language. Each event has a specific event type. This allows defining semantic queries and filtering also by users to receive and interpret only specific types of events.

#### 3.1.1. Notification events

The simplest event is a plain notification used for example to notify the CEP engine or interactive client about status changes. It contains a timestamp, which is also used to order events and is therefore of utter importance. These timestamps are also necessary for additional timing issues, because not all the available WS-BPEL and BPMN 2.0 execution engines are capable of reliably sending events on a defined point in time. In addition, the time needed for transmission of events may differ from process to process. To enable users to understand a message more easily, a message text is included in the notification event, which can be displayed on the client for explanatory or debugging purposes. Each event also contains a correlation, which relates it to the specific process and context it originated from making it possible to return results to the source of an event.

Additionally, the event itself can contain a query, which is evaluated by the CEP engine. Only if this query returns a result, the event is passed to the CEP. This offers developers of business processes the possibility to define events also depending on the current context. A timeout defined for the queries ensures that queries finish in time and do not block or invalidate events.

#### 3.1.2. Time events

The XML schema of time events with the properties described above, which is used for the monitoring of temporal properties

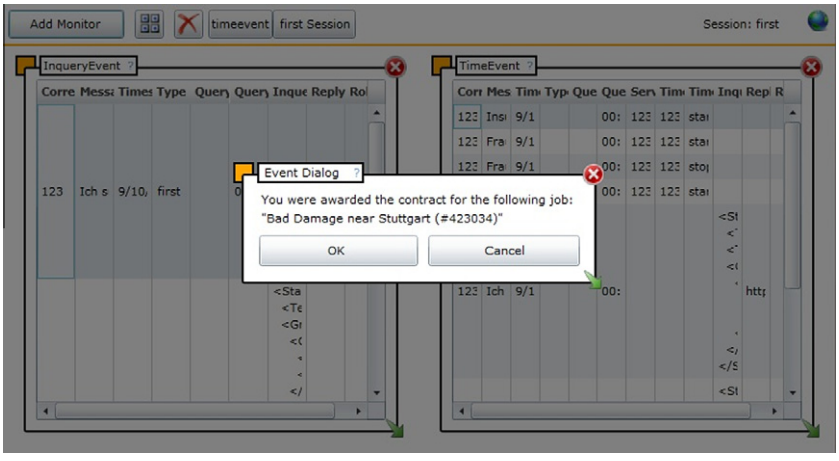


**Figure 2** Dialog interface of the client application.

of processes or as a basis for other event types, can be found below:

```
<xs:complexType name="TimeEventType">
  <xs:sequence>
    <xs:element name="Message'' type="xs:string''
/>
    <xs:element name="CorrelationID''
type="xs:string'' />
    <xs:element name="Timestamp''
type="xs:dateTime'' />
  </xs:sequence>
  <xs:attribute name="Type'' type="xs:string'' />
  <xs:attribute name="Query'' type="xs:string'' /
>
    <xs:attribute name="QueryTimeout''
type="xs:duration'' />
  </xs:complexType>
```

If time durations of processes, process steps or service calls need to be calculated, the time duration event type can be used, which specifies the start and the end of a temporal measurement:

```
<xs:complexType name="TimeDurationEventType">
  <xs:complexContent>
    <xs:extension base="mytns:TimeEventType">
      <xs:sequence>
        <xs:element name="ServiceInstanceID''
type="xs:string'' />
        <xs:element name="TimeDurationID''
type="xs:string'' />
        <xs:element name="TimeDurationType''
type="mytns:TimeDurationEnum'' />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

### 3.1.3. Interaction events

Interactions in most cases produce results. The inquiry event type is used to communicate these results or information entered by a user back to the process execution engine(s). For the prototype, we have used a simple mechanism to derive interactions from events: the client requires embedded interactions in the events, so it can directly display them for gathering information from the user. This also offers the opportunity of providing an individual interface for each type of event.

The event must also include a property that defines who is allowed to see and enter information or, more generally, use the interactions provided with the event – according to the role selection in BPEL4People or the user task in BPMN 2.0. Selection of the correct role has been implemented using XPath, so that the select-expression is being evaluated for an XML document, which exists as a profile for each user who can connect to the client application. This XML document specifies all the roles of the particular user. Its structure is also defined by an appropriate XML schema.

If results have been requested by a specific stakeholder or by a part of the system (e.g. a WS-BPEL or a BPMN 2.0 engine), a URI (Uniform Resource Identifier) is used to indicate the intended recipient of the information. At this URI, an application has to be available, which implements a defined interface for receiving the interaction results. The user interaction follows this schema:

```
<xs:complexType name="InquiryEventType">
  <xs:complexContent>
    <xs:extension base="mytns:TimeEventType">
      <xs:sequence>
        <xs:element name="InquiryForm''
type="mytns:InquiryFormType"/>
        <xs:element name="ReplyToUrl''
type="xs:string"/>
        <xs:element name="RoleQuery''
type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The user interface is defined in the inquiry form type, which contains the specification of the user interface and the return values to be sent back as a result. The event can carry user interface descriptions in any kind of specification language, which is identified by an additional attribute. As long as an interpreter, which generates the user interface from this description, exists or is newly built for this purpose, any language can be used. In addition, the parser/interpreter is responsible for reading the return values from the dialog or dashboard. These return values are defined in a way that they also describe the dialog element and its properties. If none of these values are specified, only the information contained within the inquiry is displayed to the user. The definition of the interactions reads as follows:

```
<xs:complexType name="InquiryFormType''
mixed="true">
  <xs:sequence>
    <xs:element name="Content">
      <xs:complexType>
        <xs:sequence>
          <xs:any namespace="##any''
processContents="skip'' />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ReturnControlNames">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="ReturnControlName''
type="xs:string'' minOccurs="0''
              maxOccurs="unbounded'' />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="lang'' type="xs:string'' /
>
  </xs:complexType>
```

Further events have also been defined using XML schema.

### 3.2. System architecture

Our implemented system consists of a server with a CEP engine collecting all the events generated by the different process execution engines or by other applications integrated with the

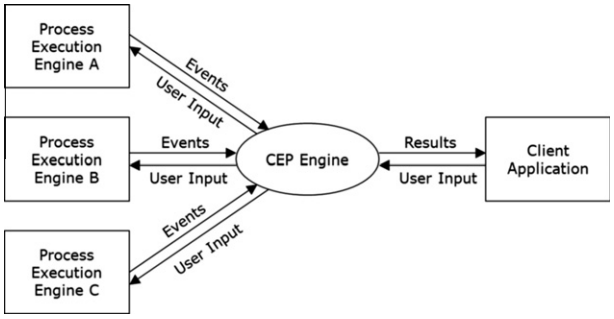


**Figure 3** System architecture.

system. As already mentioned, any WS-BPEL or BPMN 2.0 execution engine able to send events in the required form to the CEP engine may be used for process execution. These events are collected and analyzed by the CEP engine for further processing. The server also handles registration of new users and elements in the client application serving as a user interface as well as a dashboard. The system architecture is illustrated in Fig. 3.

The client application's user interface allows the definition of queries for monitoring and managing the whole platform. These queries are stored in the CEP engine which analyzes the incoming event cloud for the defined event patterns. If a query returns results, the server also deals with transferring them as emitted events to the user interfaces registered with it to achieve notifications in real-time.

From the user interface, input of users can be sent back to the process execution engines. These user inputs are also transferred via the server, which establishes a connection with the receiving process execution engine and transmits the data to it. Interactive sessions are stored by the user interface management and can be reloaded on login. The server stores the user sessions independent of existing connections, as these are likely to change. One session is also capable of supplying data to multiple user interfaces active at a certain point of time.

#### *3.2.1. Server architecture*

The CEP engine is the core of the server, collecting events from the process execution engines and passing results on to the client application. It also contains a session management component, which stores the sessions of users, including the current connections to the different user interfaces used for input from and output to the user. The CEP engine server contains interfaces to the process execution engines and other applications, which deliver events to it. This interface consists of only one operation accepting events. A big part of the application collaborates with the integrated CEP engine. The server offers a class that reacts on results of the CEP engine and passes these results on to the client application registered with it. The server architecture with the CEP engine, its session management and the mentioned interfaces is shown in Fig. 4.

#### *3.2.2. Client application*

The client application provides the user with an interface for creating new sessions and queries within them. It also displays output dialogs and accepts events from the process execution engines. The window management offers elements for displaying queries in a window-like view. The query dispatcher

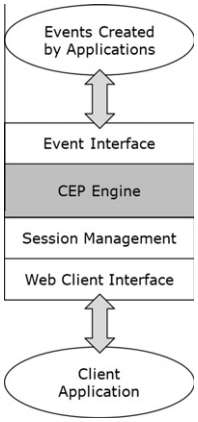


**Figure 4** Server architecture.

accepts results and passes them to the windows registered for the adjacent query, which are then displayed by the user interface in the form of visual diagrams of arriving events. A screenshot of the client application's dashboard displaying Time Event monitoring data for one specific session is shown in Fig. 5.

## 4. Technology and implementation

The architecture and implementation has to build on existing technologies in order to be applicable in practice. Therefore, we analyzed current technologies to find the most appropriate ones for our purposes.

### *4.1. CEP engine*

Esper[3] is a widely used CEP engine offering a runtime for Java and .NET. It can be integrated in any application by transmitting information via a web service call or an Enterprise Service Bus (ESB) to the CEP engine – independent of a specific process execution engine. This complies with the goal of offering a decentralized and product-independent solution. Our server contains the CEP engine Esper providing a web service interface which can be used by any process execution engine for the transmission of events.

### *4.2. Process execution engines*

Apache ODE[4] is the WS-BPEL engine of the Apache Software Foundation allowing the definition of processes in WS-BPEL 2.0. It offers fast and easy installation and deployment of business processes, but lacks some stability in the development environment. Nevertheless, it is a widely used and very useful engine for the execution of WS-BPEL processes.

Currently, two open source engines are able to natively execute BPMN 2.0, namely Activiti[5] and JBoss jBPM.[6] Both of them are still in development, but first stable versions with basic functionalities are already available and useful for testing purposes.

[3] http://esper.codehaus.org
[4] http://ode.apache.org/
[5] http://www.activiti.org
[6] http://www.jboss.org/jbpm

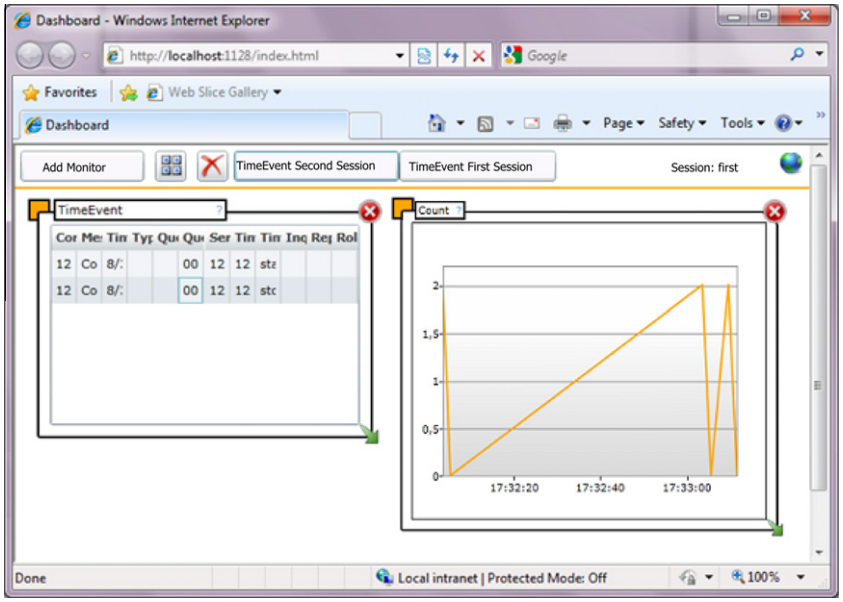


**Figure 5** Screenshot of the client application.

### *4.3. Graphical user interface language*

As already mentioned, the user interface and especially the event-driven interactions on it are defined within the events transmitted and processed via the CEP engine. Therefore, they are already generated when an event is triggered – depending on the type, context and origin of it. This means that after creation of the event, interactions are already defined. Other solutions like the use of object-oriented or semantic models for describing them (Schlegel, 2009) are more flexible – especially regarding their interpretation by the client application – but introduce additional complexity for software and process developers. Furthermore, they can cause unforeseen reactions of the user interface. Hence, we chose to use an interface description language that is directly displayable and executable by a runtime environment.

For this reason, the language had to be following standards and also allowing embedding it into the payload of an event, i.e. being integrated into its XML structure. This requires the representation of the model, which does not need to be compiled before execution in the runtime environment. Just as the well-known Hyper Text Markup Language (HTML), the eXtensible Application Markup Language (XAML) can be interpreted in a runtime environment as well, which, in this case, is based on Microsoft .NET and Silverlight.

## 5. Related work

Current research in the field of business process monitoring focuses mainly on single instances of processing engines and single sites as event sources. Baresi and Guinea proposed a system to dynamically monitor WS-BPEL processes based on certain monitoring rules, which are executed by a dedicated monitoring server (Baresi and Guinea, 2005). This approach has been further developed towards a unified framework for the monitoring and recovery of BPEL processes (Baresi et al., 2008). Hermosillo et al. talk about gathering data from business processes in order to improve them using complex event processing and dynamic business process adaptation techniques (Hermosillo et al., 2010).

In (Ammon et al., 2008) the authors detail the term "Event-Driven Business Process Management" (EDBM) as a combination of Business Process Management (BPM) for modeling and management of business processes, and Complex Event Processing (CEP). However, this approach is again limited to processes solely within one company.

Research considering monitoring cross-site processes executed by multiple WS-BPEL processors has been conducted in (Kikuchi et al., 2007). Wetzstein et al. developed an event-based system for monitoring of business processes across organizational boundaries (Wetzstein et al., 2010) with the help of BPEL4Chor service choreography descriptions (Decker et al., 2007). Here, a service choreography models the publicly visible processes and message exchanges between participants from a global viewpoint. These choreographies act as a basis for the specification of so called monitoring agreements between executing partners.

## 6. Discussion

We have successfully tested our implemented system prototype using simple automated processes from the insurance domain, as they were described in Section 1.2. Executing business processes by using mechanisms like multiple process execution engines often makes it impossible to monitor and manage the execution of these processes as a whole infrastructure. Therefore, we presented an approach and a system implementation that uses a complex event processing (CEP) engine to integrate different, cross-organizational WS-BPEL and BPMN 2.0 engines as well as other applications into a centralized common service platform, which differentiates our work from similar research activities. With regard to interactions with the processes and services running in such an infrastructure, we provided also a concept for creating ad-hoc user interactions on a monitoring

dashboard, which allows platform managers and stakeholders in the processes to interact with the platform and the processes executed – independent of their context of execution.

In contrast to related work, we are independent of a specific process execution engine and even of the process execution language (e.g. WS-BPEL or BPMN 2.0) as well as capable of using multiple engine instances. Our approach benefits especially large and decentralized system structures, as they occur when different organizations and infrastructures have to be integrated into one platform. In addition, we achieved the goal of being able to monitor such a platform and support management tasks for platform providers as well as service and process owners involved.

The concept of using embedded XAML has many advantages with regard to simplicity of changes and the reduction of efforts for creating a running system. A dynamic user interface creation is not possible when using pure XAML. Hence, we are currently evaluating HTML with JavaScript for dynamic changes on the client side.

However, from a research perspective, a model-based approach for the context-based creation of interfaces is a goal for the future. Model-interpretation or transformation could then be integrated either on the server-side, i.e. generating target code like HTML or XAML on the server, or on the client-side, i.e. sending model fragments interpreted by the client application.

In the future, decentralized and multi-engine approaches will gain higher importance due to service infrastructures spanning multiple organizations and integrating different process technologies.

## References


Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M., 2007. Web Services Human Task (WS-HumanTask), Version 1.0. Retrieved from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf.

Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., Müller, R., Pfau, G., Plösser, K., Rangaswamy, R., Rickayzen, A., Rowley, M., Schmidt, P., Trickovic, I., Yiu, A., Zeller, M., 2009. WS-BPEL Extension for People (BPEL4People), Version 1.0. Retrieved from http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., Yiu, A., 2007. Web Services Business Process Execution Language Version 2.0. Retrieved from http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

Ammon, R.v., Emmersberger, C., Springer, F., Wolff, C., 2008. Event-driven business process management and 1st practical application taking the example of DHL. In: FIS 2008/1st International Workshop on Complex Event Processing for Future Internet–Realizing Reactive Future Internet, September 28–30, 2008, Vienna, Austria

Box, D., Cabrera, L.F., Critchley, C., Curbera, F., Ferguson, D., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Niblett, P., Orchard, D., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchuk, J., Weerawarana, S., Wortendyke, D., 2006. Web Services Eventing (WS-Eventing). Retrieved from http://www.w3.org/Submission/2006/SUBM-WS-Eventing-20060315.

Baresi, L., Guinea, S., 2005. Toward dynamic monitoring of WS-BPEL processes. In: ICSOC 2005, Third International Conference of Service-Oriented Computing, volume 3826 of Lecture Notes in Computer Science.

Baresi, L., Guinea, S., Pasquale, L., 2008. Toward a unified framework for the monitoring and recovery of BPEL processes. In: TAV-WEB '08 Proceedings of the 2008 Workshop on Testing, Analysis, and Verification of Web Services and Applications.

Bruns, R., Dunkel, J., 2010. Event-Driven Architecture – Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse. Springer, Berlin, Heidelberg.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., 2003. A unifying reference framework for multi-target user interfaces. Interacting With Computers. 15 (3), 289–308.

Chandy, K.M., Schulte, W.R., 2010. Event Processing – Designing IT Systems for Agile Companies. McGraw-Hill, New York.

Decker, G., Kopp, O., Leymann, F., Weske, M., 2007. BPEL4Chor: Extending BPEL for modeling choreographies. In: IEEE International Conference on Web Services 2007 (ICWS 2007), July 3–9, 2007, pp. 296–303.

Etzion, O., Niblett, P., 2010. Event Processing in Action. Manning, Stamford, CT.

Hermosillo, G., Seinturier, L., Duchien, L., 2010. Using complex event processing for dynamic business process adaptation. In: IEEE International Conference on Service Computing 2010 (SCC 2010), July 5–10, 2010, pp.466–473.

Kikuchi, S., Shimamura, H., Kanna, Y., 2007. Monitoring method of cross-sites' processes executed by multiple WS-BPEL Processors. In: The 9th IEEE International Conference on E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007 (CEC/EEE 2007), July 23–26, 2007, pp. 55–64.

Luckham, D., 2002. The Power of Events – An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Boston, London.

Luckham, D., Schulte, R, 2008. Event Processing Glossary – Version 1.1. Retrieved from http://www.ep-ts.com/component/option,com_docman/task,doc_download/gid, 66/Itemid,84 on 8.6.2011.

Mühl, G., Fiege, L., Pietzuch, P., 2006. Distributed Event-based Systems. Springer, Berlin, Heidelberg.

OMG, 2011. Business Process Model and Notation (BPMN) – Version 2.0. Retrieved from http://www.omg.org/spec/BPMN/2.0/PDF.

Schlegel, T., 2009. Object-oriented interactive processes in decentralized production systems. In: Proceedings of the 13th International Conference on Human-Computer Interaction (HCI International 2009).

Schlegel, T., 2010. An interactive process meta model for runtime user interface generation and adaptation. In: Proceedings of the Fifth International Workshop on Model-Driven Development of Advanced User Interfaces (MDDAUI 2010) at the 28th ACM Conference on Human Factors in Computing Systems (CHI 2010).

Silver, B., 2010. BPMN Method and Style. Cody-Cassidy Press, Aptos, CA.

Vidačković, K., Kett, H., Renner, T., 2009. Event-driven service chain monitoring for the internet of services. In: Proceedings of the eChallenges e-2009 Conference.

Wetzstein, B., Karastoyanova, D., Kopp, O., Leymann, F., Zwink, D., 2010. Cross-organizational process monitoring based on service choreographies. In: SAC'10 Proceedings of the 2010 ACM Symposium on Applied Computing.